Efficient and Robust Path Openings using the Scale-Invariant Rank Operator

van de Gronde, Jasper J.; Offringa, Andre R.; Roerdink, Jos B.T.M.

Published in:
Journal of Mathematical Imaging and Vision

DOI:
10.1007/s10851-016-0649-5

J Math Imaging Vis (2016) 56:455–471
DOI 10.1007/s10851-016-0649-5

Efficient and Robust Path Openings Using the Scale-Invariant Rank Operator

Jasper J. van de Gronde[1] · **André R. Offringa**[2] · **Jos B. T. M. Roerdink**[1]

Received: 1 May 2015 / Accepted: 17 March 2016 / Published online: 19 April 2016

Abstract Some basic properties of a slightly generalized version of the scale-invariant rank operator are given, and it is shown how this operator can be used to create a nearly scale-invariant generalization of path openings that is robust to noise. Efficient algorithms are given for sequences and directed acyclic graphs with binary values, as well as sequences with real (greyscale) values. An algorithm is also given for directed acyclic graphs with real weights. It is shown that the given algorithms might be extended even further by allowing for scores based on a totally ordered semigroup.

Keywords Scale invariance · Rank operator · Path opening

1 Introduction

Rank operators are often used for filtering images, and can be considered in the framework of mathematical morphology. They are generalizations of the minimum, maximum and median filters: at each position selecting the rth value in a sorted list of all values within a certain neighbourhood. Previously, we introduced the scale-invariant rank operator (SIR operator) on binary masks, with an application in radio astronomy [22]. This is a highly efficient operator that identifies all intervals (of any size) that contain a sufficient fraction of ones in the input mask. The name stems from the fact that this operator can be interpreted as combining the results of rank operators with linear structuring elements of all possible lengths.

The SIR operator is an interesting option for anyone who needs to robustly "grow" regions or close gaps, without knowing in advance what size these regions have. Indeed, we showed [22] that this can work quite well in practice. However, the operator was limited to 1D signals (rows/columns of images were processed independently). In this work, we analyse the SIR operator in more detail, exposing a link to path openings [13,14,19]. In Sect. 3, we show how the SIR operator can be generalized to higher-dimensional signals by considering paths, leading to a straightforward generalization of path openings that allows for gaps in paths. We also briefly discuss the generalization to greyscale. In the current context, paths are understood to be approximately, but not necessarily perfectly, straight curves, formalized as paths in certain directed acyclic graphs (see Fig. 1). When a long path is broken up into smaller segments because of noise, the "missing" pixels can be said to constitute gaps. In many practical applications, it is crucial to be able to deal with gaps.

Section 4 presents algorithms for the four cases we treat: binary sequences, greyscale sequences, binary directed acyclic graphs, and greyscale directed acyclic graphs. With the exception of greyscale graphs, all cases allow an algorithm that is at most a logarithmic factor worse than linear in the size of the problem. The results are roughly in line with known results for path openings [12,14].

Section 5 compares generalized path openings with two existing schemes for making path openings robust to noise: *incomplete* path openings [14,29] and *robust* path openings [8]. Incomplete path openings allow for fine-grained tun-

✉ Jasper J. van de Gronde
j.j.van.de.gronde@rug.nl

Jos B. T. M. Roerdink
j.b.t.m.roerdink@rug.nl

[1] Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

[2] ASTRON, The Netherlands Institute for Radio Astronomy, Postbus 2, 7990 AA Dwingeloo, The Netherlands

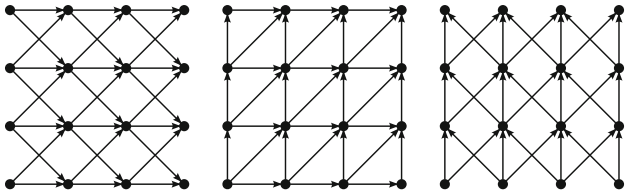

Fig. 1 Some DAGs that have been used for path openings on images

ing of how many pixels may be missing from a path, but are inefficient when this number becomes high. Robust path openings put a cap on the maximum size of each individual gap, allowing (at the strictest setting) up to 50 % of a path to be empty, while still requiring more passes over the data than a normal path opening. The generalized path openings we introduce are nearly as efficient as normal path openings, while also allowing for a fine-grained tuning of the minimum fill fraction of a path.

Section 6 shows several example applications of the generalized path openings developed in this work. Section 7 explores how much further we can generalize the SIR operator's scoring function, without requiring any fundamental changes to the algorithms.

2 Related Work

The scale-invariant rank (SIR) operator is meant as a scale-invariant version of the traditional rank operator, or rather rank-max operator[1] [15,28]. The rank operator simply returns the kth largest value within a certain window at each position. If the image domain is $\mathbb{Z}$ (the set of all integers), the (binary) image f is represented as a subset of $\mathbb{Z}$, and the window is an interval of length n starting at the current position, the rank operator can be defined as follows:

$$\zeta_{n,k}(f) = \{x \mid x \in \mathbb{Z} \text{ and } |f \cap [x, x+n-1]| \geq k\}.$$

Here $|X|$ gives the number of elements in the set X and $[a, b]$ is considered the set of all elements between a and b (including a and b). Note that in the binary case, returning the kth "largest value" is equivalent to determining whether or not there are at least k elements in the window.

Perhaps counter-intuitively, it is possible to have a position x be present in $\zeta_{n,k}(f)$, but not all of the other positions in the corresponding window. To remedy this, and make the window positioning with respect to the current position irrelevant, one can define the rank-max operator:

$$\bar{\zeta}_{n,k}(f) = \bigcup \big\{[x, x+n-1] \mid x \in \mathbb{Z} \text{ and } |f \cap [x, x+n-1]| \geq k\big\}. \quad (1)$$

[1] It is perhaps more common to consider the rank-max *opening*. This is the meet of the identity operator and the rank-max *operator*.

It is this operator that the SIR operator tries to make scale invariant.

To make a scale-invariant version of Eq. (1), the first thing to do is to allow intervals of all sizes rather than just a single size. Next, we replace the parameter k by a parameter s which is effectively k/n. This yields the original SIR operator [22][2] (note that n corresponds to $b - a + 1$)

$$\rho_s(f) = \bigcup_{n \in \mathbb{N}} \bar{\zeta}_{n,\lceil s\,n \rceil}(f) = \bigcup \big\{[a, b] \mid a, b \in \mathbb{Z},\ a \leq b \text{ and } |[a, b] \cap f| \geq s\,|[a, b]|\big\}. \quad (2)$$

Our work below concerns not just a more general SIR operator, but also a unification of the SIR operator and path openings. In this light, we will now briefly introduce path openings on graphs. Given a directed acyclic graph $G = (V, E)$, a path p is a sequence of vertices such that any pair of consecutive vertices is an edge in E. The set of vertices in p is denoted by $\sigma(p)$, its length by $|p|$, and the set of all paths in G by $\Pi(G)$. Note that in a directed acyclic graph, $|p| = |\sigma(p)|$. A path opening on a subset f of V, with path length threshold l, can then be defined as[3]

$$\alpha_{1,l}(f) = \bigcup \big\{\sigma(p) \mid p \in \Pi(G) \text{ and } \sigma(p) \subseteq f \text{ and } |p| \geq l\big\}. \quad (3)$$

The unification of path openings with the SIR operator will allow for a certain level of robustness to gaps in paths. The two main existing schemes for path openings that can deal with gaps in paths are incomplete path openings and robust path openings. Incomplete path openings [14,29] allow a path of length L with at most K missing values, with the time and space complexities linear in both the size of the graph *and* K (for the binary case). Robust path openings [8] allow for a maximum gap size, rather than a (global) maximum number of missing values. The time complexity is not entirely clear, but in practice a linear dependence on the maximum gap size is observed.

3 The New SIR Operator

Below we first introduce a slightly more general SIR operator on sequences (1D signals), and then present a further generalization to graphs (or nD signals/images) that completes the unification of the SIR operator and path openings. Note that we will mostly discuss binary signals, only briefly discussing

[2] Note that we originally defined the SIR operator with a parameter $\mu = 1 - s$.

[3] That we write $\alpha_{1,l}$ rather than α_l is to maintain consistency with the generalized operator $\alpha_{s,l}$ introduced later.

the generalization to greyscale towards the end of the current section. The implementations do support greyscale though.

To begin unifying path openings and the SIR operator, let us start by rewriting path openings so that the condition $\sigma(p) \subseteq f$ becomes integrated with the condition $|p| \geq l$, and we get closer to the condition in Eq. (2) (recall that in a directed acyclic graph $|p| = |\sigma(p)|$):

$$\begin{aligned}\alpha_{1,l}(f) &= \bigcup \big\{\sigma(p) \mid p \in \Pi(G) \\ &\qquad \text{and } \sigma(p) \subseteq f \text{ and } |p| \geq l\big\} \\ &= \bigcup \big\{\sigma(p) \mid p \in \Pi(G) \\ &\qquad \text{and } |\sigma(p) \cap f| \geq \infty \times |\sigma(p) \setminus f| + l\big\}, \end{aligned} \tag{4}$$

using the extended real numbers $\mathbb{R} \cup \{-\infty, \infty\}$. Like Alipranti and Burkinshaw [1, § 1.5], we will consider $\pm\infty \times 0 = 0$. This choice will be further motivated shortly.

Comparing Eq. (4) to Eq. (2) (giving paths the role of intervals) suggests using an inequality of the form

$$|\sigma(p) \cap f| \geq c(s)\,|\sigma(p) \setminus f| + l, \tag{5}$$

for some function $c : [0, 1] \to \mathbb{R} \cup \{\infty\}$ that goes to infinity as its parameter (s) goes to 1 from below (and is finite below 1). We now note that the limit behaviour of this inequality motivates the choice we made regarding the value to assign to $\infty \times 0$, given that $\lim_{c \uparrow \infty} c \times 0 = 0$.[4] As a result, Eq. (5) recovers path openings in the limit, while providing some robustness to gaps when $c(s)$ is finite. For example, if $c(s) = 1$ and $l = 0$, then only those intervals that have a fill fraction of at least 50 % will be kept. If l is non-zero, we recover the same behaviour in the limit as the interval size goes to infinity, while we also have the guarantee that each interval that we keep has a length of at least l.

To determine the proper form of $c(s)$, we rewrite the last part of the condition in Eq. (2):

$$\begin{aligned} &|[a, b] \cap f| \geq s\,|[a, b]| \\ &\quad \iff |[a, b] \cap f| \geq s\,(|[a, b] \cap f| + |[a, b] \setminus f|) \\ &\quad \iff (1 - s)|[a, b] \cap f| \geq s\,|[a, b] \setminus f| \\ &\quad \iff \big|\big[a, b\big] \cap f\big| \geq \frac{s}{1 - s}\,\big|\big[a, b\big] \setminus f\big|. \end{aligned}$$

It is perhaps tempting to avoid the division by $1 - s$, by not dividing both sides by $1 - s$ in the last step, but this would mean that in Eq. (6) below we would also have to multiply l by $1 - s$, leading to the inequality $0 \geq |\sigma(p) \setminus f|$ for $s = 1$; this inequality ignores l, and thus does not recover the desired behaviour. If we now combine the above ideas into one definition, we get:

Definition 1 The scale-invariant rank operator $\rho_{s,l} : \mathcal{P}(V) \to \mathcal{P}(V)$ on a graph $G = (V, E)$ is defined by

$$\begin{aligned}\rho_{s,l}(f) = \bigcup \big\{&\sigma(p) \mid p \in \Pi(G) \\ &\text{and } \big|\sigma(p) \cap f\big| \geq \frac{s}{1 - s}\,\big|\sigma(p) \setminus f\big| + l\big\}. \end{aligned} \tag{6}$$

The associated generalized path opening is defined by

$$\alpha_{s,l}(f) = f \cap \rho_{s,l}(f). \tag{7}$$

Here s is a real number greater than zero and less than or equal to one, and l a non-negative real number. If $s = 1$, we recover Eq. (4) by considering $\frac{s}{1-s}$ to equal ∞—its limit when approaching $s = 1$ from below—with the convention that $\pm\infty \times 0 = 0$ (as discussed earlier).

The parameter s gives the minimum fraction of the interval that should be set in f (at least in the limit for $|p| \to \infty$), while l gives the minimum measure of intervals in the output. Also note that instead of just counting elements, we could also use a general measure μ on $\mathbb{Z}$ [4] such that $\mu(\{x\})$ is positive for all $x \in \mathbb{Z}$. Using a general measure would allow using non-constant weights for positions, for example when dealing with non-uniform sampling. Theorem 1 below shows that $\alpha_{s,l}$ is indeed an opening. Also, it is possible to see that $\alpha_{1,l}(f)$ equals $\rho_{1,l}(f)$ (see Corollary 1) and that it recovers a traditional path opening. Combined with the fact that $\alpha_{s,l}$ can be implemented in pretty much the same way as normal path openings, these facts motivate the name.

For $l = 0$, $0 < s < 1$, and the graph $(\mathbb{Z}, \{(x, x + 1) \mid x \in \mathbb{Z}\})$, the new formulation is equivalent to the old formulation. The main advantage of this new formulation is that it allows us to have a continuum of operators all the way up to $s = 1$, in which case we get a path opening as the limit case. At first glance, it might look like the new formulation is actually less suited for use with $s = 1$ because of the division by $1 - s$, but in practice dealing with the resulting infinities is not a problem. In particular, we maintain the scores using floating point numbers, and those allow for infinity in a way that works just fine with our algorithm. If one wanted to use integers/fixed point, it would certainly be possible to do this, but some care would need to be taken to avoid overflow (or, rather, underflow).

3.1 Properties

We will now state some properties of the operator $\rho = \rho_{s,l}$. Note that the subscripts (s and l) are used only when needed for clarification.

[4] To be precise: given the specified constraints we can conclude that $|\sigma(p) \cap f| \geq \lim_{s \uparrow 1} c(s)\,|\sigma(p) \setminus f| + l \iff \sigma(p) \subseteq f$ and $|p| \geq l$, noting that neither f, nor p, nor l depends on s. This corresponds to, for our purposes, equating $\infty \times 0$ with 0.

Lemma 1 *The operator ρ is increasing: $f \subseteq g \implies \rho(f) \subseteq \rho(g)$.*

Proof Assume that $f \subseteq g$, we then prove that any element of $\rho(f)$ must also be an element of $\rho(g)$. Note that for every $x \in \rho(f)$ there is, by definition, a path $p \in \Pi(G)$ such that $x \in \sigma(p)$, $\sigma(p) \subseteq \rho(f)$, and $|f \cap \sigma(p)| \geq \frac{s}{1-s} |\sigma(p) \setminus f| + l$. Since $g \supseteq f$, we have that

$$\begin{aligned} \left|g \cap \sigma(p)\right| &\geq \left|\sigma(p) \cap f\right| \geq \frac{s}{1-s} \left|\sigma(p) \setminus f\right| + l \\ &\geq \frac{s}{1-s} \left|\sigma(p) \setminus g\right| + l. \end{aligned}$$

So, if x is in $\rho(f)$, then x is also in $\rho(g)$, which concludes the proof. □

Lemma 2 *The operator $\rho_{s,0}$ is extensive: $\rho_{s,0}(f) \supseteq f$.*

Proof Looking at Eq. (6), and noting that $\sigma(p) = \{a\}$ for every singleton path containing only $a \in V$, we see that

$$\forall a \in f : \left|f \cap \{a\}\right| = \left|\{a\}\right| \geq \frac{s}{1-s} \left|\{a\} \setminus f\right| = 0.$$

Note that, as explained previously, we consider $\frac{s}{1-s} \times 0$ to equal zero even for $s = 1$. This implies that if a is in f, then it is also in $\rho_{s,0}(f)$, proving the lemma. □

It can be seen that although any SIR operator is increasing (Lemma 1) and any $\rho_{s,0}$ is extensive (Lemma 2), SIR operators are not, in general, closings, as they are typically not idempotent. For example, taking G to be the graph whose vertices are the elements of $\mathbb{Z}$ and whose edges connect every integer to its successor,

$$\begin{aligned} &\rho_{1/2,0}(\rho_{1/2,0}(\{0\})) = [-4, 4] \\ &\quad \text{rather than } \rho_{1/2,0}(\{0\}) = [-1, 1], \end{aligned}$$

as illustrated in Fig. 2. SIR operators also do not, in general, commute with unions or intersections:

$$\begin{aligned} \rho_{1/2,0}(\{0\}) \cup \rho_{1/2,0}(\{1\}) &= [-1, 2] \\ &\neq \rho_{1/2,0}(\{0, 1\}) = [-2, 3], \text{ and} \\ \rho_{1/2,0}(\{0\}) \cap \rho_{1/2,0}(\{1\}) &= [0, 1] \neq \rho_{1/2,0}(\emptyset) = \emptyset. \end{aligned}$$

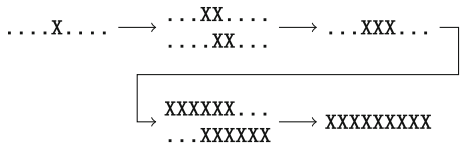

Fig. 2 A graphical depiction of what happens when $\rho_{1/2,0}$ is applied twice to an input containing a single element. From *left to right* and *top to bottom* the input, the (maximal) intervals that are found by the SIR operator, the first output (which serves as input to the second application of $\rho_{1/2,0}$), the (maximal) intervals found in the second application, and the final output

Interestingly, SIR operators *can* be made into *openings*, as shown in Theorem 1 below. Corollary 1 shows that some SIR operators are in fact openings.

The reason the SIR operator is called scale invariant is that it considers paths of all sizes, and (for $l = 0$) only looks at the fraction of the path that is present to determine whether or not to include it in the output. It thus does not depend on the scale of the signal. For $l > 0$, we lose some of this scale invariance in a controlled manner (although it is still approximately scale invariant for large intervals), which is relevant for being able to construct openings [30, Prop. 2].

3.1.1 SIR-Operator-Based Openings

Theorem 1 below shows how the SIR operator can be used to create openings, while Corollary 1 shows that some SIR operators are in fact openings.

Lemma 3 *The operator ρ is an inf-overfilter [15, ch. 12]: $\rho(f \cap \rho(f)) = \rho(f)$.*

Proof We first show that $\rho(f \cap \rho(f)) \supseteq \rho(f)$, the statement then follows from the increasing nature of ρ. To this end, note that according to Eq. (6), $\rho(f)$ is the union of all paths $p \in \Pi(G))$ satisfying $|\sigma(p) \cap f| \geq \frac{s}{1-s} |\sigma(p) \setminus f| + l$. For all such paths, we have

$$\begin{aligned} &\sigma(p) \cap f \cap \rho(f) = \sigma(p) \cap f, \text{ as well as} \\ &\sigma(p) \setminus (f \cap \rho(f)) = \sigma(p) \setminus f. \end{aligned}$$

This implies that

$$\begin{aligned} |\sigma(p) \cap f \cap \rho(f)| = |\sigma(p) \cap f| \geq \\ \frac{s}{1-s} |\sigma(p) \setminus (f \cap \rho(f))| + l = \frac{s}{1-s} |\sigma(p) \setminus f| + l, \end{aligned}$$

and thus that $\sigma(p) \subseteq \rho(f \cap \rho(f))$. Since this holds for all the intervals that make up $\rho(f)$, we have $\rho(f \cap \rho(f)) \supseteq \rho(f)$.

Finally, as ρ is increasing (Lemma 1), we have $\rho(f \cap \rho(f)) \subseteq \rho(f)$. This leads us to conclude that ρ is an inf-overfilter. □

Theorem 1 *The operator $\alpha_{s,l}$ defined by $\alpha_{s,l}(f) = f \cap \rho_{s,l}(f)$ is an algebraic opening: it is increasing, anti-extensive—$\alpha_{s,l}(f) \subseteq f$—and idempotent—$\alpha_{s,l}(\alpha_{s,l}(f)) = \alpha_{s,l}(f)$.*

Proof That $\alpha_{s,l}$ is increasing follows from the fact that $\rho_{s,l}$ is increasing (Lemma 1). That it is anti-extensive follows trivially from the definition. Its idempotency follows from the fact that $\rho_{s,l}$ is an inf-overfilter (Lemma 3), as shown in the proof of Theorem 6.26 by Heijmans [15] (alternatively: [25, Prop. 4.1]). □

Corollary 1 *The operator $\rho_{1,l}$ is equal to $\alpha_{1,l}$, and is thus an algebraic opening.*

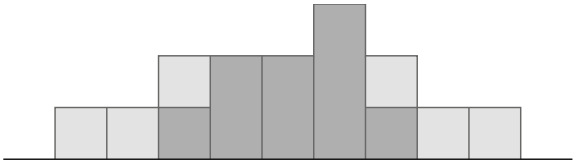

Fig. 3 Example of applying the *greyscale* SIR operator ($s = 5/7$, $l = 0$). The original (1D) signal is shown as a *bar chart* with a *dark grey fill*. *Light grey* is used to show what is added by the SIR operator. The output can be thought of as a stack of binary results. Looking just at the middle level, `.XXX.` becomes `XXXXX`, as it would in the binary case

Proof We will show that $\rho_{1,l}$ is anti-extensive, at which point it becomes clear that $\rho_{1,l} = \alpha_{1,l}$, and the claim follows immediately from Theorem 1. However, first recall that for $s = 1$, we consider $\frac{s}{1-s}\,|\sigma(p) \setminus f|$ to equal 0 if $|\sigma(p) \setminus f|$ equals zero, and infinity otherwise. Now, $|\sigma(p) \setminus f|$ is zero if and only if $\sigma(p) \subseteq f$. With these considerations, Eq. (6) takes the following form for $s = 1$:

$$\rho_{1,l}(f) = \bigcup \big\{\sigma(p) \mid p \in \Pi(G) \text{ and } \sigma(p) \subseteq f \text{ and } |\sigma(p)| \geq l\big\}.$$

In other words, it gives the union of all paths that are a subset of f, and whose length is greater than or equal to l. The union of a set of subsets of f is still a subset of f, thus we can conclude that $\rho_{1,l}$ is anti-extensive, and thus that $\rho_{1,l} = \alpha_{1,l}$, making it an algebraic opening. □

It is interesting to note that SIR operators and the generalized path openings have a relation analogous to the relation between (rank-)max operators (dilations of rank operators) and rank-max openings ([15, Ex. 6.29], [24, p. 325], [28]).

3.2 Greyscale

We now briefly discuss how to extend the problem to the greyscale case. In the greyscale case, the output is as if the binary algorithm is run with different thresholds, and each output position is set to the maximum corresponding value of the binary application (see Fig. 3). It can be checked that this recovers the interpretation of the SIR operator as a scale-invariant *rank(-max)* operator when the minimum path length $l = 0$. An example application can be the detection of radio-frequency interference (RFI) in radio astronomy: when the initial detection results in confidence levels, a greyscale SIR transform can be used directly on the confidence levels.

4 Algorithms

Here we discuss algorithms for implementing binary and greyscale versions of the SIR operator, both on sequences and on graphs. We have the following time complexities (with n the length of the sequence, V the set of vertices in the processed graph, E the set of edges, and L the set of grey levels):

- $O(n)$ for the binary operator on sequences,
- $O(|V| + |E|)$ for binary on (general) graphs,
- $O(n \log(n))$ for greyscale on sequences, and
- $O(\min(|V|, |L|)\,|V|)$ for greyscale on (sparse) graphs.

Note that we will just consider the SIR operator. Generalized path openings can be implemented by computing the intersection or infimum (position-wise minimum) of the input and the output of the SIR operator. Implementations for the greyscale SIR operator (and the generalized path opening) on sequences and images are available at http://bit.ly/1OcSZUP.

One important implementation note that concerns all algorithms is what to make of the fraction $s/(1-s)$ when $s = 1$. As mentioned earlier, our algorithms simply use floating point values, which support an actual infinity. Although this type of infinity does not support the convention $\pm\infty \times 0 = 0$, this turns out not be an issue, since the algorithms build up the scores incrementally using addition (which *is* supported in an appropriate manner).

4.1 Binary Sequences

The SIR operator can be implemented very efficiently for binary sequences. We already used this in earlier work [22], but will now provide a bit more detail. We will also present a second method that requires less temporary storage. It is also easy to implement and slightly more amenable to generalization/adaptation. Additionally, it shows how the problem is closely related to the maximum contiguous subsequence sum problem, as essentially the same algorithm can be used to solve both.

To derive our original method (adapted to the new definition of the SIR operator), we first move all terms related to counting elements to the left side of the inequality in Eq. (6) (and rephrase it in terms of intervals rather than paths):

$$\begin{aligned} &\big|[a,b] \cap f\big| \geq \frac{s}{1-s}\,\big|[a,b] \setminus f\big| + l \\ &\iff \big|[a,b] \cap f\big| + \frac{s}{s-1}\,\big|[a,b] \setminus f\big| \geq l. \end{aligned}$$

Now suppose $f : \{1, 2, \ldots, n\} \to \{0, 1\}$ is a sampled, finite length, binary signal. With the convention that $x \in f \iff f(x) = 1$, and defining $w : \{0, 1\} \to \mathbb{R}$ by $w(0) = \frac{s}{s-1}$ and $w(1) = 1$, we then get

$$\rho_{s,l}(f) = \bigcup \big\{[a,b] \mid a, b \in \mathbb{Z},\ a \leq b \text{ and } W(a,b) \geq l\big\},$$
$$\text{with } W(a,b) = \sum_{y \in [a,b]} w(f(y)).$$

So a point is output if it is part of an interval for which the score W exceeds l:

$$x \in \rho_{s,l}(f) \iff \left[\max_{a,b|a\le x\le b} W(a,b)\right] \ge l. \tag{8}$$

Note that $w(0)$ is negative for the range of s we consider: $0 < s \le 1$. Prefix sums can now be used to compute Eq. (8) efficiently:

$$x \in \rho(f) \iff \left[\max_{a,b|a\le x\le b} M(b+1) - M(a)\right] \ge l,$$
$$\text{with } M(x) = \sum_{y<x} w(f(y)).$$

Independently optimizing a and b gives

$$x \in \rho(f) \iff \left[\max_{b|b\ge x} M(b+1)\right] - \left[\min_{a|a\le x} M(a)\right] \ge l. \tag{9}$$

Clearly the prefix sums, as well as the maxima and minima in this last equation, can be easily computed for all elements in linear time using simple recurrence relations. This means that we can compute $\rho(f)$ in linear time. This is the algorithm we presented previously [22], extended to support l.

We will now present a new algorithm that is often more convenient. To this end, observe that instead of Eq. (9) we can also use

$$\begin{aligned} x \in \rho_{s,l}(f) \iff & \left[\max_{a|a\le x} \sum_{y=a}^{x} w(f(y))\right] \\ & + \left[\max_{b|b\ge x} \sum_{y=x}^{b} w(f(y))\right] - w(f(x)) \\ & = \lambda_+(x) + \lambda_-(x) - w(f(x)) \ge l. \end{aligned} \tag{10}$$

Figure 4 compares the functions used in Eqs. (8)–(10).

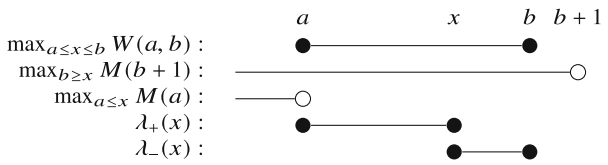

Fig. 4 Illustration of intervals involved in the different schemes for computing the maximum of the interval weight $W(a,b)$ (for $a \le x \le b$). Our previous method first computed the prefix sums for all pixels (M), and then found the optimal a and b, computing $W(a,b)$ as $M(b+1) - M(a)$ (the correct a and b are found by computing prefix minima and postfix maxima of M). Our new method first computes λ_+ and λ_-—the maximum sums to the left and right of each position—and then computes the maximum of interval weight containing x as $\lambda_+(x) + \lambda_-(x) - w(x)$

Algorithm 1: Compute λ_+.

Input : The data $f : \{1, 2, \ldots, n\} \to \{0, 1\}$.
Output: $\lambda_+ : \{1, 2, \ldots, n\} \to \mathbb{R}$ giving the maximum lengths of intervals ending at each position.

Allocate λ_+.
for $x = 1, 2, \ldots, n$ **do**
  $\lambda_+(x) \leftarrow \max(0, \lambda_+(x-1)) + w(f(x))$

Algorithm 2: Binary SIR operator.

Input : The data $f : \{1, 2, \ldots, n\} \to \{0, 1\}$.
Output: A map $\rho : \{1, 2, \ldots, n\} \to \{0, 1\}$ giving the result of the SIR operator on f.

Compute λ_+ and λ_- according to Algorithm 1 and allocate ρ.
for $x = 1, 2, \ldots, n$ **do**
  $\rho(x) \leftarrow (\lambda_+(x) + \lambda_-(x) - w(f(x)) \ge l)$

Clearly, $\lambda_+(x)$—the maximum score attainable by any interval ending at x—can be considered to be zero for $x < 1$ (before the domain of f). Also, if we know $\lambda_+(x-1)$, then $\lambda_+(x)$ can be either $\lambda_+(x-1) + w(f(x))$ or $w(f(x))$, whichever is higher (recall that $w(0)$ is negative). In particular,

$$\text{if } \arg\max_{a|a\le x} \sum_{y=a}^{x} w(f(y)) < x, \text{ then}$$
$$\max_{a|a\le x} \sum_{y=a}^{x} w(f(y)) = w(f(x)) + \max_{a|a\le x-1} \sum_{y=a}^{x-1} w(f(y)).$$

This leads to Algorithm 1 (also see Sect. 7) and an analogous algorithm for computing λ_-, which is almost exactly the same as an algorithm presented by Bentley [5, Alg. 4] for solving the maximum contiguous subsequence sum problem. The main differences are a local, rather than a global, point of view, and a disregard for empty intervals, leading to some minor differences in the code. Note that we could, but generally do not need to, keep track of the starting and ending points of the intervals (a and b in Eq. (10)).

We can now also compute $\lambda_-(x)$ on f as $\lambda_+(n+1-x)$ on f', with $f'(x') = f(n+1-x')$, and then $\rho(f)$ using $\lambda_+(x) + \lambda_-(x) - w(f(x))$ as prescribed by Eq. (10). This leads to Algorithm 2, which produces the same result as the earlier algorithm based on Eq. (9). As it sums everything twice, it might be a bit more expensive. However, if it is advantageous to trade memory for a bit of arithmetic, we can merge the computation of λ_+ or λ_- with the computation of $\rho(f)$, giving an algorithm that only needs the λ_+ array for temporary storage.

4.2 Binary Graphs

The new path-based SIR operator can be computed using an algorithm that is virtually identical to Algorithm 2, except

Algorithm 3: Compute λ_+ on a graph $G = (V, E)$.

Input : A directed acyclic graph (V, E) and image $f : V \to \mathbb{R}$.
Output: A map $\lambda_+ : V \to \mathbb{R}$ giving the maximum lengths of paths ending in each vertex.

Allocate λ_+.
for $x \in V$ *in topological order* **do**
  $\lambda_+(x) \leftarrow \max(\{0\} \cup \{\lambda_+(y) \mid (y, x) \in E\}) + w(f(x))$

that λ_+ and λ_- would be computed using Algorithm 3. This can be compared directly to the algorithm for computing a path opening transform introduced by Heijmans et al. [14, Alg. 2]. It follows that $\alpha_{1,l}(f)$ and $\rho_{1,l}(f)$ are just as efficient as traditional (binary) path openings, with a time complexity in $O(|V| + |E|)$, and amenable to many of the same tweaks and adaptations [6,16,20].

4.3 Greyscale Sequences

We will now present the greyscale version of Algorithm 1 for sequences, which has a lower time complexity than applying the general algorithm presented below to sequences. For 1D greyscale *path openings* (equivalent to attribute openings on length in 1D), a very efficient algorithm running in $O(1)$ (amortized) cost per pixel was presented by Morard et al. [19]. This was recently adapted to compute path openings on graphs as well [12] (albeit with a worse time complexity). In a nutshell, this algorithm uses a stack of path lengths, which get updated using an algorithm quite similar to Algorithm 1, except that elements are pushed onto and popped off the stack to keep track of the currently relevant grey levels. Unfortunately, this algorithm depends on only having to access the top of the stack; for the SIR operator this is not enough, as we may need to keep track of the scores of grey levels above the current grey level of the input.

In the binary case, the *SIR transform*[5] can be defined using $\lambda(x) = \lambda_+(x) + \lambda_-(x) - w(f(x))$, by saying that $x \in \rho_{s,l}(f) \iff \lambda(x) \geq l$ (see Eq. (10)). In the greyscale case, the range of f is no longer $\{0, 1\}$, but (some subset of) the reals, so we use $\lambda : V \times \mathbb{R} \to \mathbb{R}$ to give the score at each position and grey level, with V the domain of f (in general, the vertices of a graph (V, E)). We now have

$$\rho_{s,l}(f)(x) = \sup\{v \mid \lambda(x, v) \geq l\}.$$

Note that $\lambda(x, v)$ is weakly decreasing in v. Also, we again compute λ based on λ_+ and λ_-, such that

$$\lambda(x, v) = \lambda_+(x, v) + \lambda_-(x, v) - w(f(x) \geq v); \tag{11}$$

for the purposes of the last term $f(x) \geq v$ is considered to equal one if it is true, and zero otherwise.

[5] By analogy with opening transforms.

If we were to apply Algorithm 3 on all grey levels at the same time, then we would have (with $0 < s \leq 1$)

$$\lambda_+(x, v) = \max(\{0\} \cup \{\lambda_+(y, v) \mid (y, x) \in E\}) + w(f(x) \geq v), \tag{12}$$

and similarly for λ_-. The functions λ_+ and λ_- can also be described using per-position sets of (value, score)-tuples, given by $\Lambda_+, \Lambda_- : V \to \mathcal{P}(\mathbb{R} \times \mathbb{R})$, such that

$$\lambda_\pm(x, v) = \sup\left(\{w(0)\} \cup \{c \mid (v', c) \in \Lambda_\pm(x) \text{ and } v' \geq v\}\right). \tag{13}$$

Now we note that in the 1D case, Eq. (12) takes the following (simple) form:

$$\lambda_+(x, v) = \max(0, \lambda_+(x - 1, v)) + w(f(x) \geq v).$$

We can also construct a corresponding update for Λ_+:

$$\Lambda_+(x) = \begin{array}{l}\{(v, \max(0, c) + w(f(x) \geq v)) \\ \quad \mid (v, c) \in \Lambda_+(x - 1)\} \\ \cup \{(f(x), \max\{0, \lambda_+(x - 1, f(x))\} + w(1))\}.\end{array}$$

It can be checked that the following is equally valid:

$$\Lambda_+(x) = \begin{array}{l}\{(v, c + w(f(x) \geq v)) \\ \quad \mid (v, c) \in \Lambda_+(x - 1) \text{ and } c > 0\} \\ \cup \{(f(x), \max\{0, \lambda_+(x - 1, f(x))\} + w(1))\}.\end{array} \tag{14}$$

This leads to the correct λ_+ according to Eq. (13) because those tuples where c was less than or equal to zero would simply have their score set to $w(0)$ or $w(1)$ depending on whether their associated values are above or below $f(x)$, and this is already taken care of by the inclusion of $w(0)$ in the supremum in Eq. (13) and the final part of Eq. (14).

To make the above efficient, binary search trees can be used to represent Λ_+ and Λ_- (instead of the stacks used by Morard et al. [19]). This allows $O(\log(n))$ insertion, lookup, and deletion, with n the length of the sequence. So if we ignore having to update any scores, this would allow for an algorithm whose time complexity is in $O(n \log(n))$, only a logarithmic factor worse than for path openings. The rest of this section shows how to keep track of the correct scores, without sacrificing this time complexity. The overall algorithm is detailed in Algorithm 4.

4.3.1 Building Λ_+ and Λ_-

We will build a balanced binary search tree, representing $\Lambda_+(x - 1)$ (or $\Lambda_-(x - 1)$), on all values encountered so far

Algorithm 4: The algorithm for greyscale sequences. Keep in mind that the trees are implemented as functional data structures.

Input : The sequence $f : \{1, 2, \ldots, n\} \to \mathbb{R}$, and the parameters s and l.
Output: The filtered sequence $\rho(f)$.

```
// Compute Λ+(x) for all x.
```
Initialize Λ_{cur} to an empty tree.
for x *from* 1 *to* n **do**
  $\Lambda_{cur} \leftarrow \text{prune}(\Lambda_{cur})$
  $\Lambda_{cur} \leftarrow \text{update}(\Lambda_{cur}, f(x))$ `// Algorithm 6`
  $\Lambda_+(x) \leftarrow \Lambda_{cur}$

```
// Compute Λ−(x) for all x.
```
Initialize Λ_{cur} to an empty tree.
for x *from* n *to* 1 **do**
  $\Lambda_{cur} \leftarrow \text{prune}(\Lambda_{cur})$
  $\Lambda_{cur} \leftarrow \text{update}(\Lambda_{cur}, f(x))$ `// Algorithm 6`
  $\Lambda_-(x) \leftarrow \Lambda_{cur}$

```
// Compute final output
```
for x *from* 1 *to* n **do**
  $\rho(x) \leftarrow \text{maxValid}(\Lambda_+(x), \Lambda_-(x))$ `// Algorithm 7`

Algorithm 5: "Accessing" a node.

Input : A node η.

if $l(\eta)$ *is not null* **then**
  $m(l(\eta)) \leftarrow m(l(\eta)) + m(\eta)$
if $r(\eta)$ *is not null* **then**
  $m(r(\eta)) \leftarrow m(r(\eta)) + m(\eta)$
$c(\eta) \leftarrow c(\eta) + m(\eta)$
$m(\eta) \leftarrow 0$

Algorithm 6: Inserting/updating $f(x)$ into a binary search tree, ignoring balancing. Note that this relies on first having pruned the tree, so that $c(\eta) > 0$.

Input : The node η (initially the root of the tree), the value $f(x)$, and the original credit c_{suc} associated with the immediate successor (in the original tree) of the subtree rooted at η (initially 0).
Output: A new node η' reflecting the updated (sub)tree.

if η *is* **null then**
  return new node with value $f(x)$ and credit $c_{suc} + w(1)$
Access η using Algorithm 5.

```
// Update missed fields of children.
```
if $f(x) \le v(\eta)$ *and* $r(\eta)$ *is not* **null then**
  Add $w(0)$ to $m(r(\eta))$.
if $f(x) \ge v(\eta)$ *and* $l(\eta)$ *is not* **null then**
  Add $w(1)$ to $m(l(\eta))$.

```
// Recurse on the correct child (if
    necessary).
```
if $f(x) < v(\eta)$ **then**
  $l(\eta) \leftarrow \text{update}(l(\eta), f(x), c(\eta))$ `// Original c(η)`
else if $f(x) > v(\eta)$ **then**
  $r(\eta) \leftarrow \text{update}(r(\eta), f(x), c_{suc})$

```
// Only now update c(η).
```
Add $w(f(x) \ge v(\eta))$ to $c(\eta)$.

return η

for every position x. Given these trees, we can effectively reconstruct $\lambda(x, v)$ using Eqs. (13) and (11). Section 4.3.2 discusses how to do this reconstruction efficiently, while this section describes how to efficiently build all those trees. In particular, we will build these trees by scanning through the positions, and building $\Lambda_\pm(x)$ from $\Lambda_+(x-1)$ or $\Lambda_-(x+1)$. By deferring updates of scores until we access nodes anyway, and by using functional data structures, we keep the overall number of operations (as well as the space requirements) limited to $O(n \log(n))$.

Imagine that for position x we have already built a balanced binary search tree on all values encountered so far (before x), along with their associated scores, representing $\Lambda_+(x-1)$ (or $\Lambda_-(x-1)$). To build $\Lambda_+(x)$, we would (conceptually) need to copy $\Lambda_+(x-1)$, insert $f(x)$ with associated score $\max\{0, \lambda_+(x-1, f(x))\}$, and update all scores. In particular, each tuple (v, c) stored in the tree would have to have $w(f(x) \ge v)$ added to c, see Eq. (14). Clearly, this is all way too expensive, so instead of copying the tree, we use functional data structures to ensure that after inserting $f(x)$ we get a "new" tree that only differs from the original in those nodes that were affected by the insertion of $f(x)$. To see how we can also avoid updating all scores, notice that the set of nodes we do not visit is a union of subtrees rooted at children of nodes we do visit. As a result, we can keep track of what updates they missed in their roots. Each time we access a node, we use this information to update its score and to propagate the missed scores to its children. The rest of this section shows how this is done.

In the remainder, a binary search tree is a rooted binary tree in which the left subtree of each node only contains values less than the value associated with the node, and the right subtree only contains larger values. If η is a node in the tree, then $v(\eta)$ equals the value associated with that node, $c(\eta)$ equals the score associated with that node, $l(\eta)$ gives the left child, and $r(\eta)$ gives the right child. Note that after updating the tree for position x, $c(\eta)$ should equal $\lambda_\pm(x, v(\eta))$.

In more detail, to be able to efficiently keep track of the correct scores, in addition to storing a score, let every node keep track of how much score it (and its descendants) "missed" (in a separate `missed` field that is stored for each node), in the sense that those nodes should have had (an accumulated total of) so much added to their scores in previous steps (which we defer until we actually access those nodes). As soon as a node is accessed, we propagate the missed score to its direct children, we update its own score to the sum of its score and what it missed, and we set its missed score to zero (see Algorithm 5). Note that when propagating the missed score to its direct children, we do *not* recursively update any further descendants. When inserting/updating a value $f(x)$, we follow Algorithm 6. The process is illustrated in Fig. 5.

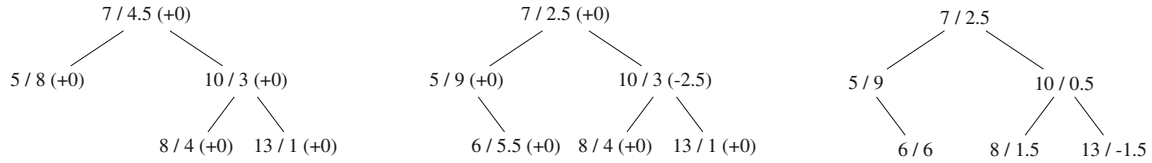

Fig. 5 The nodes in the tree on the left have labels of the form `value / score (+missed)`. If we insert 6 into the tree on the left, we get the middle tree (with $s = 5/7$, so $w(0) = -2.5$). Note that the score associated with 6 is the *original* value associated with 7 (its immediate successor in the tree) plus $w(1) = 1$. The entire right subtree is taken care of by putting -2.5 in the `missed` field of the subtree's root. Since this field describes a term that should be added to the entire subtree, conceptually we now have the tree on the right (note that the node corresponding to 13 will thus be removed if we now pruned the tree)

To now correctly implement Eq. (14), we carry out the following two steps for each position:

1. Prune the tree by removing any nodes with non-positive scores and
2. Insert the current value into the tree with a score of $w(1)$ plus the score of its immediate successor (if any), or add $w(1)$ to the old score of the current value if it was already in the tree (see Algorithm 6).

The exact details of pruning will vary depending on the search structure, but it should be possible to do this in $O(m\ \log(n))$ time per position (where m is the number of elements to delete). Since every element that is deleted was also inserted at some point, and we have at most n insertions, we see that we maintain the $O(n\ \log(n))$ overall time complexity.

It is important to notice that the extra bookkeeping only requires a constant number of operations extra per node accessed in the traditional insert/update algorithm, even if we include rotations or other manipulations that might be necessary to keep the search tree balanced. This is because it is sufficient to "access" (using Algorithm 5) the nodes involved in the rotation before performing the rotation. As a result, one can see that all operations on the tree keep their original time complexities. Our implementation uses a scapegoat tree [9]. Being a tree with amortized rather than worst-case logarithmic bounds for insertion/deletion, it is good enough for our purposes, while being simpler to implement than most other self-balancing search trees. Also, in contrast to splay trees [27], scapegoat trees still feature a worst-case logarithmic bound for lookups. This is useful for the final step: using Λ_+ and Λ_- to get the output. Note that the scapegoat tree was modified to allow storing every intermediate version, without affecting the time complexity (it was turned into a functional data structure). We expect that virtually any self-balancing search tree or skip list [21,23] can be adapted to work with our algorithm.

4.3.2 The Final Output

We still need to combine Λ_+ and Λ_- to compute the output, and preferably without affecting the time complexity. One way to do this is to simply keep track of the trees encoding Λ_+ and Λ_- at each position, and then to search these trees simultaneously for the right output value. Assuming that the trees are balanced (their height is in $O(\log(n))$), this can be done in $O(\log(n))$ time per pixel.

Once we have $\Lambda_+(x)$ and $\Lambda_-(x)$ (represented by binary search trees of finite, non-zero, size), notice that

$$\begin{aligned}\lambda(x, v) &= \lambda_+(x, v) + \lambda_-(x, v) - w(f(x) \geq v)\\ &= \max\left(\{w(0)\} \cup \{c \mid (v', c) \in \Lambda_+(x) \text{ and } v' \geq v\}\right)\\ &\quad + \max\left(\{w(0)\} \cup \{c \mid (v', c) \in \Lambda_-(x) \text{ and } v' \geq v\}\right)\\ &\quad - w(f(x) \geq v).\end{aligned} \tag{15}$$

As a result, $\rho_{s,l}(f)(x)$ can be found as the highest grey level for which $\lambda(x, v) \geq l$. Searching for this value in a single tree would be almost trivial, because $\lambda(x, v)$ is non-increasing in v (so the tree can be used to look up nodes both on v and on the associated score). Alternatively, if the trees had the same shape, we could clearly search the trees in parallel. Unfortunately, the authors are unaware of an algorithm that would allow merging the two binary search trees in $O(\log(n))$ time, but below we show we can still efficiently search the trees in parallel, even if their shapes do not match.

Based on the definition of λ given in Eq. (15), we can define a function that determines the highest value v for which $\lambda(x, v) \geq l$, given two trees A and B (corresponding to Λ_+ and Λ_-):

$$\begin{aligned}&maxValid(A, B)\\ &\quad = \max\left\{v \,\middle|\, (v, \cdot) \in A \cup B \text{ and } \lambda(x, v) \geq l\right\}.\end{aligned}$$

Here the trees are treated as sets of pairs. If no valid result exists, $maxValid$ should return some "bottom" value for which the condition always holds (like 0 or $-\infty$).

We will now demonstrate how to efficiently compute the result of $maxValid$. To this end, identify a tree A with its root element, such that $v(A)$ equals the value associated with the root element for example. We now introduce

$$c(A, B) = c(A) + c(B) - w(f(x) \geq \max\{v(A), v(B)\}).$$

Consider $c(A)$ equal to $\lambda_+(x, v(A))$ and $c(B) = \lambda_-(x, v(B))$. Although $c(A, B)$ in general does not equal either $\lambda(x, v(A))$ or $\lambda(x, v(B))$, recalling that $\lambda(x, v)$ is weakly decreasing in v, we do have Lemma 4.

Lemma 4 *Given two subtrees A and B of trees encoding Λ_+ and Λ_- (not necessarily in that order), we have*

$$\lambda(x, \max(v(A), v(B))) \leq c(A, B) \leq \lambda(x, \min(v(A), v(B))).$$

Proof The first inequality follows from

$$\lambda_+(x, \max(v(A), v(B))) + \lambda_-(x, \max(v(A), v(B))) \leq c(A) + c(B).$$

The second inequality is more challenging to prove, but we do see that

$$\lambda_+(x, \min(v(A), v(B))) + \lambda_-(x, \min(v(A), v(B))) \geq c(A) + c(B).$$

Now, if both $f(x) \geq v(A)$ and $f(x) \geq v(B))$, the contributions from w are equal in both $c(A, B)$ and $\lambda(x, \min(v(A), v(B)))$, and the inequality holds. If $f(x) < v(A)$ and $f(x) < v(B)$, the contributions of w are also equal, and the inequality also holds. If one value is at or below $f(x)$ and the other is above $f(x)$, then $\lambda(x, \min(v(A), v(B)))$ subtracts $w(1)$, while $c(A, B)$ subtracts $w(0)$. At first glance, this appears problematic (since this works against the inequality), but given that in this case we have $\min(v(A), v(B)) \leq f(x) < \max(v(A), v(B))$, we see that $\lambda_\pm(x, \min(v(A), v(B))) + w(0) - w(1)$ is bounded from below by $\lambda_\pm(x, \max(v(A), v(B)))$. As a result, assuming without loss of generality that $v(A) < v(B)$, when $v(A) \leq f(x) < v(B)$

$$\begin{aligned} c(A, B) &= c(A) + c(B) - w(f(x) \geq \max(v(A), v(B))) \\ &= \lambda_+(x, v(A)) + \lambda_-(x, v(B)) - w(0) \\ &\leq \lambda_+(x, v(A)) \\ &\quad + \left[\lambda_-(x, v(A)) + w(0) - w(1)\right] - w(0) \\ &= \lambda_+(x, v(A)) + \lambda_-(x, v(A)) - w(1) \\ &= \lambda(x, \min(v(A), v(B))). \end{aligned}$$

□

Having established that Lemma 4 holds, it is now possible to give a recursive procedure for computing $maxValid$. If $l \leq c(A, B)$, then $maxValid(A, B) \geq \min(v(A), v(B))$. As a consequence, assuming $v(A) \leq v(B)$, $maxValid(A, B) = \max(v(A), maxValid(r(A), B))$. Note that this works because we are absolutely sure that $maxValid(A, B)$ will not need to consider values less than or equal to $v(A)$ to determine the validity of values higher than $v(A)$. On the other hand, if $c(A, B) < l$, then we do know that $v(B)$ is too high, *but* its score may still be required in determining the validity of lower values. So we can only say that $maxValid(A, B)$ equals $maxValid(A, l(B) \cup \{(v(B), c(B))\})$. Note that instead of actually inserting the root of B into its left subtree, we can just keep track (per side) of the last node that *should* have been inserted. Each such node is guaranteed to have a higher value than the node ultimately returned, and a lower value than the previous one we remembered for the same side, so just remembering the last one is sufficient. With this modification in place, this procedure recurses until one of the (sub)trees becomes empty. When this happens, the empty tree can be considered to have an entry with the value of the root of the other tree, with the score set to the last remembered score for that side. When both trees are empty, $maxValid$ should return the lowest possible value (like 0 or $-\infty$, and represented by $\perp$ in the algorithm). This is captured in Algorithm 7.

Algorithm 7: Computing maxValid. Note that in practice we can keep track of the minimum of v_B and v_A rather than both values separately.

Input : Two trees A and B, the values v_A^+ and v_B^+ associated with the immediate successors (in the original trees) of the current subtrees (initially ∞), and the associated scores c_A^+ and c_B^+ (initially $w(0)$).
Output: The maximum value v in A and B for which $\lambda(x, v) \geq l$.

```
// Base case.
if A is null and B is null then
    return ⊥
// Normalize input.
Access A and B using Algorithm 5.
if A is null or v(A) > v(B) then
    Swap A and B, as well as c_A^+ and c_B^+, and v_A^+ and v_B^+.
// Compute c(A, B).
if B is null then
    c_AB ← c(A) + c_B^+ − w(f(x) ≥ max{v(A), v_B^+})
else
    c_AB ← c(A) + c(B) − w(f(x) ≥ max{v(A), v(B)})
// Recurse.
if l ≤ c_AB then
    return max{v(A), maxValid(r(A), B, v_A^+, v_B^+, c_A^+, c_B^+)}
else
    return max{v(A), maxValid(A, l(B), v_A^+, v(B), c_A^+, c(B)}
```

The procedure described in the previous paragraph allows us to search both $\Lambda_-(x)$ and $\Lambda_+(x)$ in parallel, in time linearly dependent on the sum of the heights of the trees representing $\Lambda_-(x)$ and $\Lambda_+(x)$. If we assume that the trees are balanced (as in our implementation), the final computation of $\rho_{s,l}(f)$ from Λ_- and Λ_+ takes $O(n \log(n))$ time, resulting in the same overall time complexity for the entire filter. Turning the SIR operator $\rho_{s,l}$ into the generalized path opening $\alpha_{s,l}$ takes linear time, as it only requires computing the meet of the input and the result of the SIR operator.

Algorithm 8: Update of λ_+.

Input : A directed acyclic graph (V, E), an image $f : V \to \mathbb{R}$, the current grey level v, and $\lambda_+ : V \to \mathbb{R}$ for the previous (next higher) grey level.
Output: λ_+ for the current grey level v.

```
// Pixels are visited in topological order.
```
Initialize the priority queue Q with all pixels at level v.
while Q *not empty* **do**
  Remove first pixel x from Q.
  $\lambda \leftarrow \max(\{0\} \cup \{\lambda_+(y) \mid (y, x) \in E\}) + w(f(x) \geq v)$
  if $\lambda > \lambda_+(x)$ **then**
    $\lambda_+(x) \leftarrow \lambda$
    Push successors of y onto Q.

Note that in addition to using the algorithm above on sequences, one should also be able to use the above algorithm with the technique introduced by Morard et al. [20] to compute the so-called "parsimonious path openings" on 2D images (and directed acyclic graphs in general). The idea here is to apply a 1D filter to a cleverly selected [3] subset of paths, giving an approximation to the full path-based filter.

4.4 Greyscale Graphs

Our implementation of the greyscale SIR operator on 2D images (and the associated generalized path opening) is based on the algorithm for traditional greyscale path openings developed by [2,29]. This algorithm starts by computing λ_+ and λ_- for the lowest grey level, and then proceeds to update it for the next higher grey level, and so on and so forth, keeping track for each pixel of the highest level at which λ is large enough. The main differences compared to our implementation are that we chose to iterate over the grey values from high to low, and that we cannot assume that pixels outside the current upper level set do not participate. Algorithm 8 gives an overview of the most critical section of the algorithm, based on the version of Appleton and Talbot's algorithm presented by [12,26].

It should be noted that the time complexity of the greyscale algorithm on graphs is typically superlinear, and potentially quadratic (depending on the image content). This can be derived in more or less the same way as for normal path openings [12,26], except that now path weights do not need to be integers, resulting in an upper bound of $O(\min(|V|, |L|)\,|V|)$ (assuming that the grey levels can be sorted in time linear in $|L|$ and that the graph is *sparse*). Here L is the set of grey levels in the image. Note that it remains to be seen whether this bound is tight. In this light, it is interesting to see, as is shown in the previous section, that the one-dimensional case can be solved in a slightly different way, resulting in a lower (for high bit-depth images at least) time complexity of $O(|V| \log(|V|))$. It remains to be seen whether a similar approach can also be applied to general graphs, but it looks like this would at the very least severely complicate matters, as we would somehow need to (lazily) merge binary search trees very efficiently.

5 Comparison to Robust and Incomplete Path Openings

Given that generalized path openings are robust to gaps in paths, it is interesting to compare them to some other alternatives that have the same aim. Table 1 shows how generalized path openings compare qualitatively to incomplete and robust path openings when applied to small examples. We are particularly interested in how different generalized path openings are from incomplete path openings. If the two give very similar results in practice, then generalized path openings would be preferable in the vast majority of cases, as they are more efficient. Robust path openings would then only really be interesting if one is interested in finding very sparse paths, while having a useful upper bound on the maximum gap size. Towards this goal, we can see that if we have an incomplete path opening $\gamma_{L,K}$, then we can find a generalized path opening $\alpha_{s,l}$ that will be less active than the incomplete path

Table 1 A demonstration of the differences between the various types of path openings

Path pattern	Fill fraction	Generalized $l = 3,\ s = 5/7$		Incomplete $L = 7,\ K = 2$	Robust $L = 7,\ G = 1$
`X...XXX`	4/7	No		No	No
`X.X.X.X`	4/7	No		No	Yes
`X..XXXX`	5/7	No		Yes	No
`X.X.XXX`	5/7	No	$(5 \not\geq \frac{5}{2} \times 2 + 3 = 8)$	Yes	Yes
`XXXXX...XXXXXX`	11/14	Yes	$(11 \geq \frac{5}{2} \times 3 + 3 = 10.5)$	No	No
`X.X.X.XXXXXXXX`	11/14	Yes		No	Yes
`XXX..XXX`	6/8	Yes		Yes	No
`X.XXXXX`	6/7	Yes		Yes	Yes

Each row shows a pattern and whether this pattern is preserved as a single "path". Every possible combination is shown, with a couple of numerical examples for the generalized path opening

opening, in the sense that $\gamma_{L,K}(f) \leq \alpha_{s,l}(f) \leq f$. This can be seen by checking that a generalized path opening is the union of an infinite series of incomplete path openings. To this end, observe that

$$\begin{aligned}\alpha_{s,l}(f) = f \cap \bigcup \big\{ &\sigma(p) \mid p \in \Pi(G) \\ &\text{and } \left|\sigma(p) \cap f\right| \geq \frac{s}{1-s}\left|\sigma(p) \setminus f\right| + l \big\} \\ = \bigcup_{M \in \mathbb{Z}, M \geq l} \bigcup \big\{ &\sigma(p) \cap f \mid p \in \Pi_M(G) \\ &\text{and } \left|\sigma(p) \cap f\right| \geq \frac{s}{1-s}\left|\sigma(p) \setminus f\right| + l \big\}. \end{aligned} \tag{16}$$

Here $\Pi_M(G) = \{p \mid p \in \Pi(G) \text{ and } |p| = M\}$ is the set of paths in G of length M. Some manipulation shows that in Eq. (16) (note that $M = |p| = |\sigma(p)|$ by construction)

$$\begin{aligned} &\left|\sigma(p) \cap f\right| \geq \frac{s}{1-s}\left|\sigma(p) \setminus f\right| + l \\ &\quad \iff \left|\sigma(p)\right| - \left|\sigma(p) \setminus f\right| \geq \frac{s}{1-s}\left|\sigma(p) \setminus f\right| + l \\ &\quad \iff M - l \geq \frac{s + (1-s)}{1-s}\left|\sigma(p) \setminus f\right| \\ &\quad \iff M - l \geq \frac{1}{1-s}\left|\sigma(p) \setminus f\right| \\ &\quad \iff (1-s)(M-l) \geq \left|\sigma(p) \setminus f\right|. \end{aligned}$$

We can now conclude that

$$\alpha_{s,l}(f) = \bigcup_{M \in \mathbb{Z}, M \geq l} \gamma_{M, \lfloor (1-s)(M-l) \rfloor}(f). \tag{17}$$

If we solve $(1-s)(L-l) = K$ for s, we recover $s = (L-K-l)/(L-l)$. As a result, a generalized path opening $\alpha_{s,l}$ with $s = (L-K-l)/(L-l)$, assuming $0 \leq l \leq L-K$, will be less active than the incomplete path opening $\gamma_{L,K}$.

The question is what do those other incomplete path openings in Eq. (17) add (so those with $M \neq L$)? In practice, they add a lot less than might be expected, since we usually just look at the final output, not the intervals/paths that were found. This means that it usually does not really matter whether something is preserved as a single path or as multiple paths. So although the generalized path opening will preserve more paths as single paths, a lot of those would actually also be preserved by the incomplete path opening as a string of (possibly overlapping) paths. Also, as Cokelaer et al. [8] observed, it is often natural to let K scale with L.

An additional advantage of generalized path openings over incomplete path openings is that if we wish to allow for arbitrary non-integer vertex/edge weights, incomplete path openings probably become intractable. This is because they rely on a dynamic programming solution that relies on the path lengths and gap sizes being integers.

Compared to robust path openings, SIR path openings allow one to be much more strict in terms of the fraction of the path that can be missing, at the expense of having no explicit upper bound on the size of a single gap. Incomplete path openings to some extent allow both (in that a certain number of pixels can be missing for a fixed path length), but SIR path openings have the potential to be much faster. In principle, SIR path openings should hardly be slower to compute than normal path openings, while the running times of incomplete path openings scale linearly with the number of allowed missing pixels [29].

6 Examples

In this section, we show some examples of what can be done with the algorithms developed in the preceding sections. In our original work [22] on the SIR operator, we already showed what could be done with the binary operator on sequences, so we skip this and start out with the binary operator on graphs. Code to generate all the examples is available at http://bit.ly/1OcSZUP.

6.1 Binary Graphs/2D Images

Figure 6 shows the results of applying the binary generalized path opening to a test image. The parameters were selected by trial and error to give reasonable results, see our previous work [22] for an example of how suitable parameters can be found in an actual application (for the 1D SIR operator).

6.2 Greyscale Sequences

Figure 7 shows some examples of the SIR operator on sequences. The operator was applied to interferometric astronomical radio observations recorded with the Westerbork Synthesis Radio Telescope at low frequency (150 MHz). The image is a dynamic spectrum, where brightness represents the amplitude of the complex correlation between two antennas, with time on the x-axis and frequency on the y-axis. Before sky maps can be made from these data, automated RFI detectors have to remove the RFI, which are line-like features with increased brightness. As was shown earlier [22], it is advantageous to scale-invariantly extend and fill the holes in these features. Ideally, the greyscale operator should be applied on detection confidence levels, but available RFI detectors do not output these. Therefore, we use the raw data before RFI detection.

Note that even though the data are 2D in nature, it is beneficial to apply the 1D operator on each row rather than the 2D operator, as the 2D operator (in this case) suffers from what can only be described as "leakage" even for fairly high values of s (see Fig. 8), while the 1D operator can be useful even for much lower values of s, which our previous work on the SIR operator [22] showed to be useful in this setting.

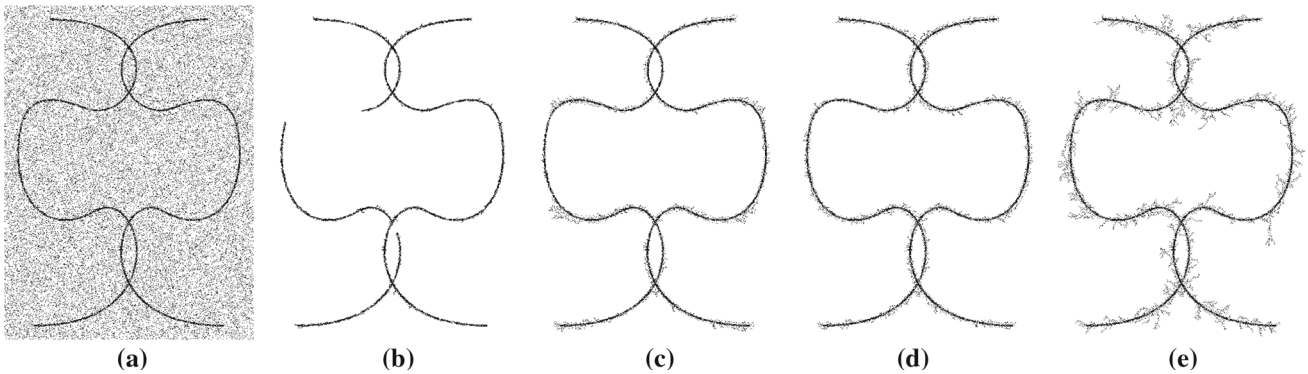

Fig. 6 **a** The original (binary) image (387 × 517 pixels), followed by the results of **b** a normal path opening $l = 100$, **c** a generalized path opening ($l = 100$, $s = 0.97$), **d** an incomplete path opening ($L = 102$, $K = 2$), and **e** a robust path opening ($L = 100$, $G = 1$). The results of the generalized path opening and the incomplete path opening are very similar, except that the generalized path opening is slightly stricter in regions where it can only find short paths (in some of the vertically oriented segments) and more lax in regions where it finds long paths (near the *top* and *bottom* of the sidewards bends). The robust path opening is much less selective

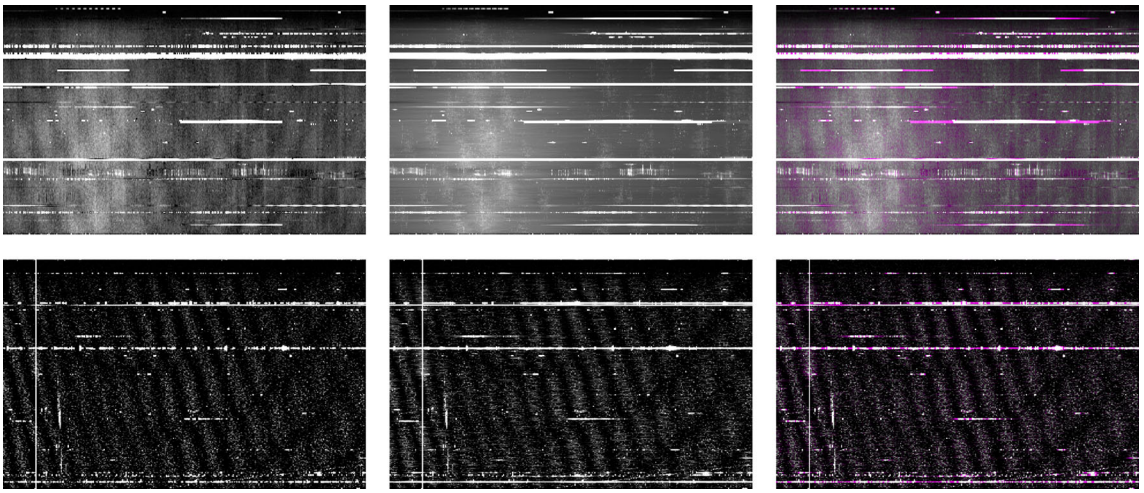

Fig. 7 Dynamic spectra (*horizontal* time, *vertical* frequency) of two astronomical interferometric radio observations. Line-like features are caused by interference, and for detection it is advantageous to scale-invariantly extend these. *Left* two radio astronomy datasets with resolutions of 379 × 256 (top) and 314 × 512 (*bottom*, a small section of the 4314 × 512 original). Note that for the second example, the filter was in fact applied to the full resolution version. *Middle* the result of applying the greyscale 1D SIR operator on them (per row, $s = 0.7$, $l = 0$). *Right* magenta shows where the SIR operator added something. The last image is made by assigning the SIR result to the *red* and *blue* channels, and the original data to the green channel. The *grey* levels are rescaled so that in the top row full scale corresponds to 0.045 Jy and in the *bottom row* full scale corresponds to 0.019 Jy (Color figure online)

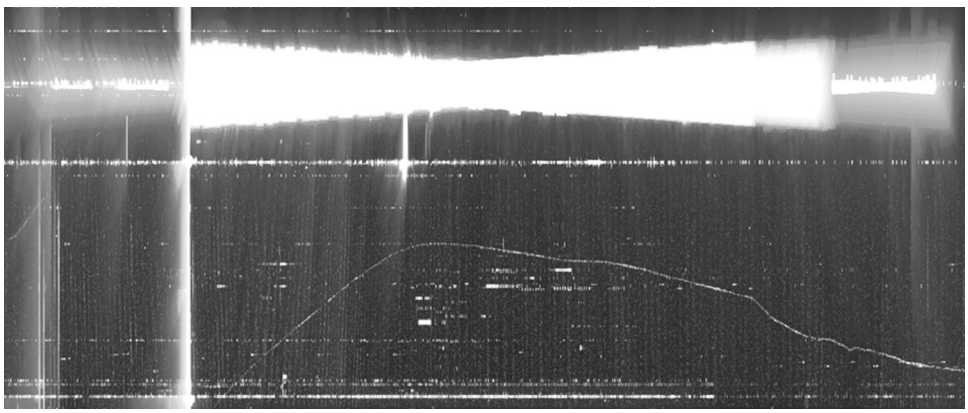

Fig. 8 The second example shown in Fig. 7, but now shown at the full 4314 × 512 resolution and filtered using the 2D SIR operator ($s = 0.98$, $l = 0$). Note how the vertical stripes suffer from "tail-like" artefacts, caused by connections to the large *horizontal stripes* in the *diagonal* graphs (see Fig. 1). Further constraints on the ability of paths to "zig–zag" might be beneficial [11,16]

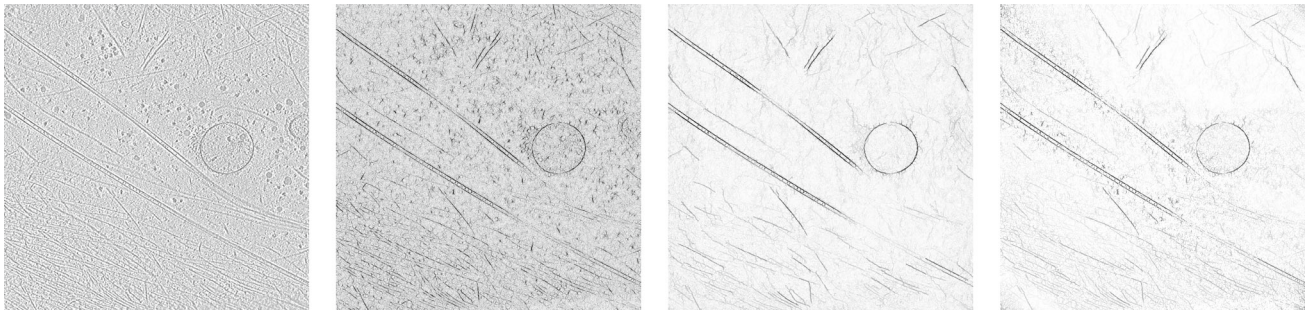

Fig. 9 From *left to right* the original image (2000 × 2000) showing a slice from a transmission electron microscopy dataset with fibroblasts and/or microtubules (provided by Robert van Liere), the fractional anisotropy map of the orientation score, a path opening ($s = 1, l = 200$), and a generalized path opening ($s = 0.95, l = 200$), both with the RORPO modification [17,18]. The contrast has been inverted (and in the original the values are scaled so that maximum fractional anisotropy corresponds to full scale)

Most likely the 2D results could probably be improved using the constraints introduced by Luengo Hendriks [16] or by integrating orientation scores [10,11], but we have not (yet) integrated these options into our implementation.

6.3 Greyscale Graphs/2D images

Figure 9 shows an application to 2D slices from a transmission electron microscopy dataset. From the original greyscale data, another greyscale image is created by first computing a second-order tensorial orientation score [10], and then encoding the fractional anisotropies of the tensors as grey levels. The fractional anisotropy of a 2-by-2 symmetric matrix equals $\sqrt{(\lambda_1 - \lambda_2)^2}/\sqrt{\lambda_1^2 + \lambda_2^2}$, where λ_1 and λ_2 are eigenvalues. Typically, this measure is only computed for (symmetric) positive semidefinite matrices, but it is still meaningful for matrices with negative eigenvalues, and the overwhelming majority of the matrices (over 98 %) in this example are in fact positive semidefinite. This preprocessing makes the linear features stand out more from the background, as in the original dataset the range of grey levels is roughly similar both in the background and along the features, making it very difficult for a (generalized) path opening to preserve the linear features while suppressing the background.

The "Ranking Orientation Responses of Path Operators" (RORPO) modification [17,18] was incorporated into our implementation of generalized path openings to further reduce the background response, since this modification responds only to long *and* thin structures. This is accomplished by applying the path opening separately to the different graphs corresponding to different directions (see Fig. 1), and then computing the final result as the difference between the highest and the fourth-highest (in this case the lowest) response for each pixel.

It is clear that both results suppress a lot of background noise, while retaining much of the original line-like features.

Algorithm 9: Binary SIR operator (generalized).

Input : A directed acyclic graph (V, E) and data-dependent weights $w_f : V \to \mathcal{S}$.
Output: A map $\rho : V \to \{0, 1\}$ giving the result of the SIR operator on f.

Allocate $\lambda_+ : V \to \mathcal{S}$.
for $x \in V$ *in topological order* **do**
  $\lambda_+(x) \leftarrow \max\left(\{w_f(x)\} \cup \{\lambda_+(y) \bullet w_f(x) \mid (y, x) \in E\}\right)$
Allocate $\lambda_- : V \to \mathcal{S}$.
for $x \in V$ *in reverse topological order* **do**
  $\lambda_-(x) \leftarrow \max\left(\{w_f(x)\} \cup \{w_f(x) \bullet \lambda_-(y) \mid (x, y) \in E\}\right)$
Allocate $\rho : V \to \mathcal{S}$.
for $x \in V$ **do**
  $\rho(x) \leftarrow \max\left(\{\lambda_-(x)\} \cup \{\lambda_+(y) \bullet \lambda_-(x) \mid (y, x) \in E\}\right)$

The generalized path opening can be seen to preserve a bit more of some of the features (especially in the right half of the image, below the circle), but interestingly it also suppresses more of the background (above the circle for example). This is because by itself it will tend to preserve more of the background in each direction, making the RORPO modification more effective.

7 Generalized Scoring Functions

The SIR operator is quite attractive in its simplicity and efficiency (especially in the binary case), and in this work we have already generalized it slightly compared to our original formulation, without requiring any significant changes to the algorithm. In this section, we will explore the limits of what more can be done without requiring a radically different algorithm.

We first present Algorithm 9: a generalized equivalent of combining Algorithm 3 and Algorithm 2. Here $\mathcal{S}$ is a totally ordered set, and '$\bullet$' is an associative binary operator on this totally ordered set. The overall time complexity is still $O(|V| + |E|)$, with $|V|$ being the number of vertices

and $|E|$ the number of edges in the input graph. Requiring '$\bullet$' to be associative guarantees that we can assign an unambiguous score to any path, regardless of the traversal order. Therefore, we will assume that '$\bullet$' is associative, and the score $S(abc\ldots)$ of the sequence of weights $abc\ldots$ will be equated to $a \bullet b \bullet c \bullet \cdots$. Note that the computation of λ_- has been modified (compared to that for λ_+) to give the same score as λ_+ for the same path, using the associativity of '$\bullet$'.

We now need a criterion for when the algorithm is correct. This is conceptually simple: $\rho(x)$ should contain the highest possible score associated with any path through x. Formally,

$$\rho(x) = \max\{S(w_f(p)) \mid p \in \Pi(V, E) \text{ and } x \in \sigma(p)\}.$$

Here $w_f(p)$ denotes the sequence of (data-dependent) weights corresponding to the vertices in the path p, and $\Pi(V, E)$ denotes the set of all paths in the directed acyclic graph (V, E).

Lemma 5 *A sufficient condition for Algorithm 9 to be correct is*

$$a \leq b \implies a \bullet c \leq b \bullet c \text{ and } c \bullet a \leq c \bullet b \tag{18}$$

for all $a, b, c \in \mathcal{S}$.

Proof To see that this condition is indeed sufficient, we first show that right before computing ρ, $\lambda_+(x)$ contains the highest possible score of any path ending at x. To this end, imagine that there would be some y for which $\lambda_+(y)$ is too low (it can never be too high, as the algorithm effectively computes the overall score for a particular path ending at y). There must then be some $x \neq y$ from which y can be reached, such that at x, choosing a lower than maximal score would have been better for the score at y. However, this cannot be, since if we equate c with the score of the path from x to y (not including x), Eq. (18) guarantees that taking the maximum score at x results in the highest score at y as well. In the same way, λ_- can be shown to contain the highest possible scores for all paths beginning in each vertex. We are now in a position to show that ρ is computed correctly.

Again, ρ cannot be too high, so imagine it is too low. This means that we either should have taken a different value for $\lambda_-(x)$, and/or for one of the $\lambda_+(y)$ ($(y, x) \in E$). Since λ_+ and λ_- contain the highest possible scores for paths ending/beginning in each vertex, it is immediately clear from the condition above that using different values (corresponding to different paths) can only decrease the final result, so ρ cannot possibly be too low. This concludes the proof. □

Although Lemma 5 gives a *sufficient* condition, it may not be necessary. Still, there is not a lot of room for weakening the condition. For example,

$$[a \bullet b \leq b \text{ and } c \leq b \bullet c] \implies a \bullet b \bullet c \leq b \bullet c$$

can already be seen to be a necessary condition. Here $a \bullet b$ plays the role of a in Eq. (18), and we have an extra condition on c. That the above condition is necessary can be shown by simply constructing a sequence of three vertices (numbered one to three) with the weights $w_f(1) = a$, $w_f(2) = b$, and $w_f(3) = c$, such that $a \bullet b \leq b$ and $c \leq b \bullet c$. We can then verify that if the implication does not hold, we have $\lambda_+(3) = b \bullet c < a \bullet b \bullet c$. So far, we have not been able to find a satisfactory set of necessary *and* sufficient conditions. However, even if they can be found, one may wonder how useful they would be in inspiring new scoring schemes, as they would likely be fairly specific.

Summarizing, our algorithm works for any scheme based on an associative operator '$\bullet$' operating on a totally ordered set $\mathcal{S}$, such that Eq. (18) holds; in other words, if $\mathcal{S}$ is a totally ordered semigroup with operator '$\bullet$' [7]. Note that there might be other schemes that also work, but they would still need to satisfy something similar to Eq. (18). Also, it might be possible to make further changes to the algorithm to allow for a wider range of scoring schemes, or even to allow for a slightly milder condition on '$\bullet$' than associativity, but this is beyond the current scope.

This just leaves the question of whether anything interesting is possible, beyond what we have shown here. Defining $a \bullet b = \max(a, b)$ would definitely work, and so would a scheme using a lexicographical order on tuples of numbers with per-position addition (vectors). Whether these options will find an application, and what other options exist, remains to be seen.

8 Conclusion

We have presented a generalized version of the SIR operator we previously introduced, and determined that although the SIR operator itself is not an opening or closing, it *is* a so-called inf-overfilter. This allowed us to construct openings (and closings) based on the SIR operator that are approximately scale invariant and generalize path openings.

Interestingly, our generalized path openings give an efficient and scale-invariant method for finding long paths with gaps. This method has a lower time complexity than at least incomplete path openings [14,29], and likely robust path openings [8] as well, while having the same kind of flexibility as incomplete path openings. The main thing that sets our method apart from both earlier methods is that it is approximately scale invariant, and thus does not put any cap on the maximum gap size, as long as the gap is a small enough fraction of the complete path. Whether and when this is an advantage or a disadvantage remains to be seen.

We also looked at generalizing the SIR operator to greyscale. Compared to the binary algorithm, the greyscale algorithm on sequences is fairly complex, but its time com-

plexity of $O(n \log(n))$ is still quite reasonable, and we have provided a prototype implementation. This algorithm is heavily inspired by the algorithm given by [19] for 1D (path) openings. Like for path openings, the greyscale-graph-case appears to be much more difficult to implement efficiently than both the greyscale-sequence-case and the binary-graph-case. We provide an implementation for greyscale images (/graphs) that is roughly of equal cost as the existing algorithms for traditional path openings [16,29].

In our examples, we have shown a very simple way to combine greyscale generalized path openings with orientation scores [10,11]. This approach is not ideal, but provides a simple way of incorporating at least some information on (the local "strength" of the) orientation. In future work, it would be interesting to incorporate orientation scores more fully into the algorithm, for example by computing the path openings in different directions on different orientations in the orientation score. What makes this particularly interesting is that both orientation scores and generalized path openings are (approximately) scale invariant, and that the two algorithms complement each other: orientation scores provide local information on orientation, while generalized path openings find global structures.

Since especially the binary algorithms presented here are very attractive in terms of simplicity and speed, we have examined how much further we can generalize the scoring functions. It appears that there is still some room for other types of scoring functions, as long as they are based on associative operators satisfying an order-preservation condition. It will be interesting to see if any applications can be found where these other types of scoring functions will prove useful. Finally, we wonder whether it is possible to give a statistical argument for using the scoring function given here, or find other path scoring functions which are somehow optimal from a statistical point of view, while still allowing for efficient computation.

Acknowledgments The authors would like to thank Robert van Liere for making available some interesting and high-quality datasets. This research is (partially) funded by The Netherlands Organisation for Scientific Research (NWO), Project No. 612.001.001.

References

1. Aliprantis, C.D., Burkinshaw, O.: Principles of Real Analysis. Academic Press, San Diego (1998)
2. Appleton, B., Talbot, H.: Efficient Path Openings and Closings. In: Ronse, C., Najman, L., Decencière, E. (eds.) Mathematical Morphology: 40 Years On Computational Imaging and Vision, vol. 30, pp. 33–42. Springer, The Netherlands (2005). doi:10.1007/1-4020-3443-1_4
3. Asplund, T.: Improved Path Opening by Preselection of Paths. Master's thesis, Uppsala Universitet (2015)
4. Bartle, R.G.: The Elements of Integration. Wiley, New York (1966)
5. Bentley, J.: Programming pearls: algorithm design techniques. Commun. ACM **27**, 865–873 (1984). doi:10.1145/358234.381162
6. Bismuth, V., Vaillant, R., Talbot, H., Najman, L.: Curvilinear Structure Enhancement with the Polygonal Path Image: Application to Guide-Wire Segmentation in X-Ray Fluoroscopy. In: Ayache, N., Delingette, H., Golland, P., Mori, K. (eds.) Medical Image Computing and Computer Assisted Intervention. LNCS, vol. 7511, pp. 9–16. Springer, Berlin (2012). doi:10.1007/978-3-642-33418-4_2
7. Clifford, A.H.: Totally ordered commutative semigroups. Bull. Am. Math. Soc. **64**(6), 305–316 (1958). doi:10.1090/S0002-9904-1958-10221-9
8. Cokelaer, F., Talbot, H., Chanussot, J.: Efficient robust d-dimensional path operators. IEEE J. Sel. Top. Signal Process. **6**(7), 830–839 (2012). doi:10.1109/jstsp.2012.2213578
9. Galperin, I., Rivest, R.L.: Scapegoat trees. In: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms. SODA '93, pp. 165–174. Society for Industrial and Applied Mathematics, Philadelphia (1993)
10. van de Gronde, J.J.: Tensorial Orientation Scores. In: Azzopardi, G., Petkov, N. (eds.) Computer Analysis of Images and Patterns. LNCS, vol. 9257, pp. 783–794. Springer International Publishing, Switzerland (2015). doi:10.1007/978-3-319-23117-4_67
11. van de Gronde, J.J., Lysenko, M., Roerdink, J.B.T.M.: Path-Based Mathematical Morphology on Tensor Fields. In: Hotz, I., Schultz, T. (eds.) Visualization and Processing of Higher Order Descriptors for Multi-Valued Data, Mathematics and Visualization, pp. 109–127. Springer International Publishing, Switzerland (2015a). doi:10.1007/978-3-319-15090-1_6
12. van de Gronde, J.J., Schubert, H.R., Roerdink, J.B.T.M.: Fast Computation of Greyscale Path Openings. In: Benediktsson, J.A., Chanussot, J., Najman, L., Talbot, H. (eds.) Mathematical Morphology and Its Applications to Signal and Image Processing. LNCS, vol. 9082, pp. 621–632. Springer International Publishing, Switzerland (2015). doi:10.1007/978-3-319-18720-4_52
13. Heijmans, H., Buckley, M., Talbot, H.: Path-based morphological openings. In: IEEE International Conference on Image Processing, vol. **5**, pp. 3085–3088. (2004). doi:10.1109/icip.2004.1421765
14. Heijmans, H., Buckley, M., Talbot, H.: Path openings and closings. J. Math. Imaging Vis. **22**(2), 107–119 (2005). doi:10.1007/s10851-005-4885-3
15. Heijmans, H.J.A.M.: Morphological Image Operators. Academic Press, San Diego (1994)
16. LuengoHendriks, C.L.: Constrained and dimensionality-independent path openings. IEEE Trans. Image Process. **19**(6), 1587–1595 (2010). doi:10.1109/tip.2010.2044959
17. Merveille, O., Talbot, H., Najman, L., Passat, N.: Tubular Structure Filtering by Ranking Orientation Responses of Path Operators. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) Computer Vision—ECCV 2014. LNCS, pp. 203–218. Springer International Publishing, Switzerland (2014). doi:10.1007/978-3-319-10605-2_14
18. Merveille, O., Talbot, H., Najman, L., Passat, N.: Ranking Orientation Responses of Path Operators: Motivations, Choices and Algorithmics. In: Benediktsson, J.A., Chanussot, J., Najman, L., Talbot, H. (eds.) Mathematical Morphology and Its Applications to Signal and Image Processing. LNCS, vol. 9082, pp. 633–644. Springer International Publishing, Switzerland (2015). doi:10.1007/978-3-319-18720-4_53
19. Morard, V., Dokládal, P., Decencière, E.: One-dimensional openings, granulometries and component trees in O(1) per pixel. IEEE

J. Sel. Top. Signal. Process. **6**(7), 840–848 (2012). doi:10.1109/jstsp.2012.2201694
20. Morard, V., Dokládal, P., Decencière, E.: Parsimonious path openings and closings. IEEE Trans. Image Process. **23**(4), 1543–1555 (2014). doi:10.1109/tip.2014.2303647
21. Munro, J.I., Papadakis, T., Sedgewick, R.: Deterministic Skip Lists. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '92, pp. 367–375. Society for Industrial and Applied Mathematics, Philadelphia (1992)
22. Offringa, A.R., van de Gronde, J.J., Roerdink, J.B.T.M.: A morphological algorithm for improving radio-frequency interference detection. Astron. Astrophys. **539**, A95+ (2012). doi:10.1051/0004-6361/201118497
23. Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. Commun. ACM **33**(6), 668–676 (1990). doi:10.1145/78973.78977
24. Ronse, C.: Order-configuration functions: mathematical characterizations and applications to digital signal and image processing. Inf. Sci. **50**(3), 275–327 (1990). doi:10.1016/0020-0255(90)90014-2
25. Ronse, C., Heijmans, H.J.A.M.: The algebraic basis of mathematical morphology. CVGIP: Image Underst. **54**(1), 74–97 (1991). doi:10.1016/1049-9660(91)90076-2
26. Schubert, H., van de Gronde, J.J., Roerdink, J.B.T.M.: Efficient computation of greyscale path openings. Math. Morphol. Theory Appl. (2015). Accepted
27. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM **32**(3), 652–686 (1985). doi:10.1145/3828.3835
28. Soille, P.: On morphological operators based on rank filters. Pattern Recognit. **35**(2), 527–535 (2002). doi:10.1016/s0031-3203(01)00047-4
29. Talbot, H., Appleton, B.: Efficient complete and incomplete path openings and closings. Image Vis. Comput. **25**(4), 416–425 (2007). doi:10.1016/j.imavis.2006.07.021
30. Urbach, E.R., Roerdink, J.B.T.M., Wilkinson, M.H.F.: Connected shape-size pattern spectra for rotation and scale-invariant classification of gray-scale images. IEEE Trans. Pattern Anal. Mach. Intell. **29**(2), 272–285 (2007). doi:10.1109/tpami.2007.28

Jasper J. van de Gronde studied computing science at the University of Groningen, the Netherlands, where he obtained his M.Sc. in 2011. He recently (June 2015) defended his Ph.D. thesis on the topic of non-scalar mathematical morphology at the University of Groningen, and is currently a post-doc at the Scientific Visualization and Computer Graphics group of the Johann Bernoulli Institute for Mathematics and Computer Science in Groningen. His research interests include mathematical morphology, compressed sensing, and signal/image processing in general.

André R. Offringa obtained his Ph.D. in 2012 from the University of Groningen, on the topic of "Algorithms for Radio Interference Detection and Removal". After his Ph.D., he worked on the Murchison Widefield Array interferometric telescope at the Mount Stromlo Observatory, a research school of the Australian National University in Canberra, to make a statistical detection of the Epoch of Reionization (EoR). Currently, he is working as a post-doc at the Netherlands Institute for Radio Astronomy (ASTRON). He is involved in a project using the LOFAR telescope to analyse the EoR. His contributions include various tools for radio astronomy, including AOFlagger, a platform to mitigate radio-frequency interference from interferometric or single-dish observations, and a fast interferometric imager called wsclean.

Jos B. T. M. Roerdink studied biology and physics at the University of Nijmegen, the Netherlands, where he obtained his M.Sc. in theoretical physics in 1979. Following his Ph.D. (1983) from the University of Utrecht and a two-year position (1983–1985) as a Postdoctoral Fellow at the University of California, San Diego, both in the area of stochastic processes, he joined the Centre for Mathematics and Computer Science in Amsterdam, where he worked from 1986 to 1992 on image processing and tomographic reconstruction. He worked as an associate professor (1992) and a full professor (2003), respectively, at the Johann Bernoulli Institute for Mathematics and Computer Science of the University of Groningen, where he currently holds a chair in Scientific Visualization and Computer Graphics. His research interests include mathematical morphology, biomedical visualization, neuroimaging, and bioinformatics.